



# Distributed Knight

## Building Distributed Collaboration Tools using Type-Based Publish/Subscribe

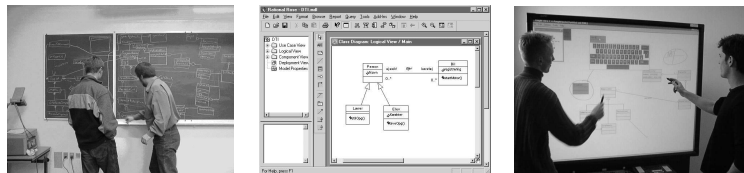
Christian Heide Damm & Klaus Marius Hansen  
University of Aarhus, Denmark



# Stand-Alone Knight



- Goal: To support collaborative, creative, and lightweight UML diagramming



# Distributed Knight



- Goal: to support collaborative, lightweight modelling in a distributed setting
  - different geographical locations
  - different types of devices
  - changes in location over time

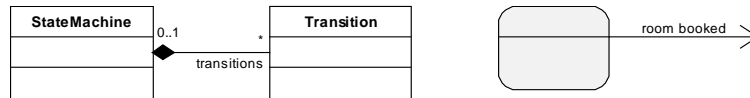


# Demo

# Implementation



- Knight implements the UML metamodel



- Data is serialized using the XMI standard
  - XML format
  - one XML tag for each element
  - compositions in UML map to nested XML tags

```
<Behavioral_Elements.State_Machines.StateMachine xmi.id="idstateMachine0-3D021CA3">
  ...
  <Behavioral_Elements.State_Machines.StateMachine.transitions>
    <Behavioral_Elements.State_Machines.Transition xmi.id="idtransition0-3D021CA3">
      <Foundation.Core.ModelElement.name>room booked</Foundation.Core.ModelElement.name>
      ...
    </Behavioral_Elements.State_Machines.Transition>
  </Behavioral_Elements.State_Machines.StateMachine.transitions>
</Behavioral_Elements.State_Machines.StateMachine>
```

# Implementation: Distributed Knight



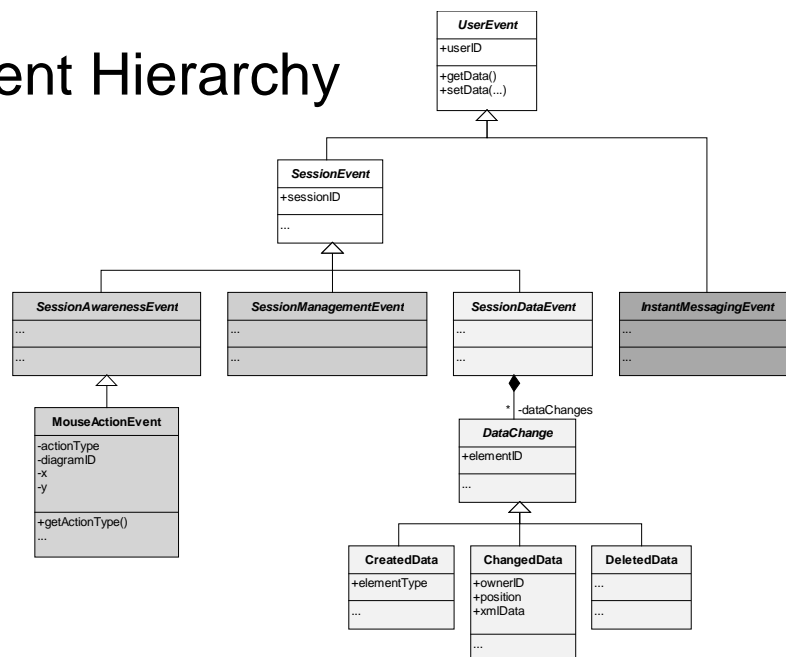
- Clients collaborate in sessions
- Data is replicated at each client in the session
- Data changes communicated using
  - type-based publish/subscribe (TPS)
  - RMI

# Type-Based Publish/Subscribe



- Event bus
  - publish events
  - subscribe to events by giving criteria
- A high-level variant of publish/subscribe
  - events are instances of application-defined types, i.e., objects
  - generalisation of subject-based and content-based publish/subscribe

## Event Hierarchy



## Advantages of TPS: Event Hierarchy



- Object-oriented
  - lets you model the "communication primitives"
- Natural successor to nested subjects in subject-based publish/subscribe
  - not necessarily strictly hierarchical
- Not Knight-specific
  - may evolve into a general, reusable distribution library

## Advantages of TPS: Stay Native



- Allows developers to continue using native types and objects
  - as opposed to shifting to more primitive communication based on name/value pairs
  - faster to develop distributed applications
  - the result is easier to understand and maintain
  - compare RMI
- Also easier to specify content filters
  - filters can be specified using native language
  - no need for a special subscription language
- Fits well with RMI

## Disadvantages of TPS



- Heterogeneous environment
  - staying native not always possible
  - data may be serialized in a standard way
  - methods are more difficult
- Evolution
  - how to handle different versions of the same class?

## Position



- Realistic experiments are necessary to evaluate and develop communication abstractions
- TPS (with RMI) is good for building distributed collaboration tools